IN THE UNITED STATES DISTRICT COURT

FOR THE NORTHERN DISTRICT OF CALIFORNIA

ORACLE AMERICA, INC.,

Plaintiff,

v.

GOOGLE INC.,

Defendant.

______________________________/

No. C 10-03561 WHA

ORDER RE COPYRIGHTABILITY OF CERTAIN REPLICATED ELEMENTS OF THE JAVA APPLICATION PROGRAMMING INTERFACE

INTRODUCTION

This action was the first of the so-called "smartphone war" cases tried to a jury. This order includes the findings of fact and conclusions of law on a central question tried simultaneously to the judge, namely the extent to which, if at all, certain replicated elements of the structure, sequence and organization of the Java application programming interface are protected by copyright.

PROCEDURAL HISTORY

In 2007, Google Inc., announced its Android software platform for mobile devices. In 2010, Oracle Corporation acquired Sun Microsystems, Inc., and thus acquired Sun's interest in the popular programming language known as Java, a language used in Android. Sun was renamed Oracle America, Inc. Shortly thereafter, Oracle America (hereinafter simply "Oracle") sued defendant Google and accused its Android platform as infringing Oracle's Java-related copyrights and patents.

Both Java and Android are complex platforms. Both include "virtual machines," development and testing kits, and application programming interfaces, also known as APIs. Oracle's copyright claim involves 37 packages in the Java API. Copyrightability of the elements replicated is the only issue addressed by this order.

Due to complexity, the Court decided that the jury (and the judge) would best understand the issues if the trial was conducted in phases. The first phase covered copyrightability and copyright infringement as well as equitable defenses. The second phase covered patent infringement. The third phase would have dealt with damages but was obviated by stipulation and verdicts.

For the first phase, it was agreed that the judge would decide issues of copyrightability and Google's equitable defenses and that the jury would decide infringement, fair use, and whether any copying was de minimis. Significantly, all agreed that Google had not literally copied the software but had instead come up with its own implementations of the 37 API packages. Oracle's central claim, rather, was that Google had replicated the structure, sequence and organization of the overall code for the 37 API packages.

For their task of determining infringement and fair use, the jury was told it should take for granted that the structure, sequence and organization of the 37 API packages as a whole *was* copyrightable. This, however, was not a final definitive legal ruling. One reason for this instruction was so that if the judge ultimately ruled, after hearing the phase one evidence, that the structure, sequence and organization in question was not protectable but was later reversed in this regard, the court of appeals might simply reinstate the jury verdict. In this way, the court of appeals would have a wider range of alternatives without having to worry about an expensive retrial. Counsel were so informed but not the jury.

Each side was given seventeen hours of "air time" for phase one evidence (not counting openings, closings or motion practice). In phase one, as stated, the parties presented evidence on copyrightability, infringement, fair use, and the equitable defenses. As to the compilable code for the 37 Java API packages, the jury found that Google infringed but deadlocked on the follow-on question of whether the use was protected by fair use. As to the documentation for the 37 Java API packages, the jury found no infringement. As to certain small snippets of code, the jury found only one was infringing, namely, the nine lines of code called "rangeCheck." In phase two, the jury found no patent infringement across the board. (Those patents, it should be noted, had nothing to do with the subject addressed by this order.) The entire jury portion of the trial lasted six weeks.[1]

This order addresses and resolves the core premise of the main copyright claims, namely, whether the elements replicated by Google from the Java system were protectable by copyright in the first place. No law is directly on point. This order relies on general principles of copyright law announced by Congress, the Supreme Court and the Ninth Circuit.

* * *

Counsel on both sides have supplied excellent briefing and the Court wishes to recognize their extraordinary effort and to thank counsel, including those behind the scenes burning midnight oil in law libraries, for their assistance.

SUMMARY OF RULING

So long as the specific code used to implement a method is different, anyone is free under the Copyright Act to write his or her own code to carry out exactly the same function or specification of any methods used in the Java API. It does not matter that the declaration or method header lines are identical. Under the rules of Java, they *must be identical* to declare a method specifying the *same* functionality — even when the implementation is different. When there is only one way to express an idea or function, then everyone is free to do so and no one can monopolize that expression. And, while the Android method and class names could have been different from the names of their counterparts in Java and still have worked, copyright protection never extends to names or short phrases as a matter of law.

It is true that the very same functionality could have been offered in Android without duplicating the exact command structure used in Java. This could have been done

[1] After the jury verdict, the Court granted Oracle's Rule 50 motion for judgment as a matter of law of infringement of eight decompiled computer files, which were literally copied. Google admitted to copying eight computer files by decompiling the bytecode from eight Java files into source code and then copying the source code. These files were not proven to have ever been part of Android.

by re-arranging the various methods under different groupings among the various classes and packages (even if the same names had been used). In this sense, there were many ways to group the methods yet still duplicate the same range of functionality.

But the names are more than just names — they are symbols in a command structure wherein the commands take the form

java.package.Class.method()

Each command calls into action a pre-assigned function. The overall name tree, of course, has creative elements but it is also a precise command structure — a utilitarian and functional set of symbols, each to carry out a pre-assigned function. This command structure is a system or method of operation under Section 102(b) of the Copyright Act and, therefore, cannot be copyrighted. Duplication of the command structure is necessary for interoperability.

STATEMENT OF FINDINGS

1. JAVA AND ANDROID.

Java was developed by Sun, first released in 1996, and has become one of the world's most popular programming languages and platforms.[2] The Java platform, through the use of a virtual machine, enables software developers to write programs that are able to run on different types of computer hardware without having to rewrite them for each different type. Programs that run on the Java platform are written in the Java language. Java was developed to run on desktop computers and enterprise servers.[3]

The Java language, like C and C++, is a human-readable language. Code written in a human-readable language — "source code" — is not readable by computer hardware.

[2] For purposes of this order, the term "Java" means the Java platform, sometimes abbreviated to "J2SE," which includes the Java development kit (JDK), javac compiler, tools and utilities, runtime programs, class libraries (API packages), and the Java virtual machine.

[3] Rather than merely vet each and every finding and conclusion proposed by the parties, this order has navigated its own course through the evidence and arguments, although many of the proposals have found their way into this order. Any proposal that has been expressly agreed to by the opposing side, however, shall be deemed adopted (to the extent agreed upon) even if not expressly adopted herein. It is unnecessary for this order to cite the record for all of the findings herein. In the findings, the phrase "this order finds . . ." is occasionally used to emphasize a point. The absence of this phrase, however, does not mean (and should not be construed to mean) that a statement is not a finding. All declarative fact statements set forth in the order are factual findings.

Only “object code,” which is not human-readable, can be used by computers. Most object code is in a binary language, meaning it consists entirely of 0s and 1s. Thus, a computer program has to be converted, that is, compiled, from source code into object code before it can run, or “execute.” In the Java system, source code is first converted into “bytecode,” an intermediate form, before it is then converted into binary machine code by the Java virtual machine.

The Java language itself is composed of keywords and other symbols and a set of pre-written programs to carry out various commands, such as printing something on the screen or retrieving the cosine of an angle. The set of pre-written programs is called the application programming interface or simply API (also known as class libraries).

In 2008, the Java API had 166 “packages,” broken into more than six hundred “classes,” all broken into over six thousand “methods.” This is very close to saying the Java API had 166 “folders” (packages), all including over six hundred pre-written programs (classes) to carry out a total of over six thousand subroutines (methods). Google replicated the exact names and exact functions of virtually all of these 37 packages but, as stated, took care to use different code to implement the six thousand-plus subroutines (methods) and six-hundred-plus classes.

An API is like a library. Each package is like a bookshelf in the library. Each class is like a book on the shelf. Each method is like a how-to-do-it chapter in a book. Go to the right shelf, select the right book, and open it to the chapter that covers the work you need. As to the 37 packages, the Java and Android libraries are organized in the same basic way but all of the chapters in Android have been written with implementations different from Java but solving the same problems and providing the same functions. Every method and class is specified to carry out precise desired functions and, thus, the “declaration” (or “header”) line of code stating the specifications must be identical to carry out the given function.[4]

The accused product is Android, a software platform developed by Google for mobile devices. In August 2005, Google acquired Android, Inc., as part of a plan to develop a smartphone platform. Google decided to use the Java language for the Android platform.

[4] The term “declaration” was used throughout trial to describe the headers (non-implementing code) for methods and classes. While “header” is the more technically accurate term, this order will remain consistent with the trial record and use “declaration” and “header” interchangeably.

In late 2005, Google began discussing with Sun the possibility of taking a license to use and to adapt the entire Java platform for mobile devices. They also discussed a possible co-development partnership deal with Sun under which Java technology would become an open-source part of the Android platform, adapted for mobile devices. Google and Sun negotiated over several months, but they were unable to reach a deal.

In light of its inability to reach agreement with Sun, Google decided to use the Java language to design its own virtual machine via its own software and to write its own implementations for the functions in the Java API that were key to mobile devices. Specifically, Google wrote or acquired its own source code to implement virtually all the functions of the 37 API packages in question. Significantly, all agree that these implementations — which account for 97 percent of the lines of code in the 37 API packages — are different from the Java implementations. In its final form, the Android platform also had its own virtual machine (the so-called Dalvik virtual machine), built with software code different from the code for the Java virtual machine.

As to the 37 packages at issue, Google believed Java application programmers would want to find the same 37 sets of functionalities in the new Android system callable by the same names as used in Java. Code already written in the Java language would, to this extent, run on Android and thus achieve a degree of interoperability.

The Android platform was released in 2007. The first Android phones went on sale the following year. Android-based mobile devices rapidly grew in popularity and now comprise a large share of the United States market. The Android platform is provided free of charge to smartphone manufacturers. Google receives revenue through advertisement whenever a consumer uses particular functions on an Android smartphone. For its part, Sun and Oracle never successfully developed its own smartphone platform using Java technology.

All agree that Google was and remains free to use the Java language itself. All agree that Google's virtual machine is free of any copyright issues. All agree that the six-thousand-plus method implementations by Google are free of copyright issues. The copyright issue, rather, is whether Google was and remains free to replicate the names, organization of those names, and functionality of 37 out of 166 packages in the Java API, which has sometimes been referred to in this litigation as the "structure, sequence and organization" of the 37 packages.

The Android platform has its own API. It has 168 packages, 37 of which are in contention. Comparing the 37 Java and Android packages side by side, only three percent of the lines of code are the same. The identical lines are those lines that specify the names, parameters and functionality of the methods and classes, lines called "declarations" or "headers." In particular, the Android platform replicated the same package, method and class names, definitions and parameters of the 37 Java API packages from the Java 2SE 5.0 platform. This three percent is the heart of our main copyright issue.

A side-by-side comparison of the 37 packages in the J2SE 5.0 version of Java versus in the Froyo version of Android shows that the former has a total of 677 classes (plus interfaces) and 6508 methods wherein the latter has 616 and 6088, respectively. Twenty-one of the packages have the same number of classes, interfaces and methods, although, as stated, the method implementations differ.

The three percent of source code at issue includes "declarations." Significantly, the rules of Java dictate the precise form of certain necessary lines of code called declarations, whose precise and necessary form explains why Android and Java *must be* identical when it comes to those particular lines of code. That is, since there is only one way to declare a given method functionality, everyone using that function must write that specific line of code in the same way. The same is true for the "calls," the commands that invoke the methods. To see why this is so, this order will now review some of the key rules for Java programming. This explanation will start at the bottom and work its way upward.

2. THE JAVA LANGUAGE AND ITS API — IMPORTANT DETAILS.

Java syntax includes *separators* (*e.g.*, {, }, ;), *operators* (*e.g.*, +, -, *, /, <, >), *literal values* (*e.g.*, 123, 'x', "Foo"), and *keywords* (*e.g.*, if, else, while, return). These elements carry precise predefined meanings. Java syntax also includes *identifiers* (*e.g.*, String, java.lang.Object), which are used to name specific values, fields, methods, and classes as described below.

These syntax elements are used to form statements, each statement being a single command executed by the Java compiler to take some action. Statements are run in the sequence written. Statements are commands that tell the computer to do work.

A method is like a subroutine. Once declared, it can be invoked or "called on" elsewhere in the program. When a method is called on elsewhere in the program or in an application, "arguments" are usually passed to the method as inputs. The output from the method is known as the "return." An example is a method that receives two numbers as inputs and returns the greater of the two as an output. Another example is a method that receives an angle expressed in degrees and returns the cosine of that angle. Methods can be much more complicated. A method, for example, could receive the month and day and return the Earth's declination to the sun for that month and day.

A method consists of the method header and the method body. A method header contains the name of the method; the number, order, type and name of the parameters used by the method; the type of value returned by the method; the checked exceptions that the method can throw; and various method modifiers that provide additional information about the method. At the trial, witnesses frequently referred to the method header as the "declaration." This discrepancy has no impact on the ultimate analysis. The main point is that this header line of code introduces the method body and specifies very precisely its inputs, name and other functionality. Anyone who wishes to supply a method with the same functionality must write this line of code in the same way and must do so no matter how different the implementation may be from someone else's implementation.

The method body is a block of code that then implements the method. If a method is declared to have a return type, then the method body must have a statement and the statement must include the expression to be returned when that line of code is reached. During trial, many witnesses referred to the method body as the "implementation." It is the method body that does the heavy lifting, namely the actual work of taking the inputs, crunching them, and returning an answer. The method body can be short or long. Google came up with its own implementations for the method bodies and this accounts for 97 percent of the code for the 37 packages.

Once the method is written, tested and in place, it can be called on to do its work. A method call is a line of code *somewhere else*, such as in a different program that calls on (or invokes) the method and specifies the arguments to be passed to the method for crunching. The method would be called on using the command format "java.package.Class.method()" where () indicates the inputs passed to the method. For example, a = java.package.Class.method() would set the field "a" to equal the return of the method called. (The words "java.package.Class.method" would in a real program be other names like "java.lang.Math.max"; "java.package.Class.method" is used here simply to explain the format.)

After a method, the next higher level of syntax is the class. A class usually includes fields that hold values (such as pi = 3.141592) and methods that operate on those values. Classes are a fundamental structural element in the Java language. A Java program is written as one or more classes. More than one method can be in a class and more than one class can be in a package. All code in a Java program must be placed in a class. A class declaration (or header) is a line that includes the name of the class and other information that define the class. The body of the class includes fields and methods, and other parameters.

Classes can have subclasses that "inherit" the functionality of the class itself. When a new subclass is defined, the declaration line uses the word "extends" to alert the compiler that the fields and methods of the parent class are inherited automatically into the new subclass so that only additional fields or methods for the subclass need to be declared.

The Java language does not allow a class to extend (be a subclass of) more than one parent class. This restrictiveness may be problematic when one class needs to inherit fields and methods from two different non-related classes. The Java programming language alleviates this dilemma through the use of "interfaces," which refers to something different from the word "interface" in the API acronym. An interface is similar to a class. It can also contain methods. It is also in its own source code file. It can also be inherited by classes. The distinction is that a class may inherit from more than one interface whereas, as mentioned, a class can only inherit from one other class.

For convenience, classes and interfaces are grouped into "packages" in the same way we all group files into folders on our computers. There is no inheritance function within packages; inheritance occurs only at the class and interface level.

Here is a simple example of source code that illustrates methods, classes and packages. The italicized comments on the right are merely explanatory and are not compiled:

```
package java.lang;                              // Declares package java.lang
public class Math {                             // Declares class Math
        public static int max (int x, int y) {  // Declares method max
                if (x > y) return x ;           // Implementation, returns x or
                else return y ;                 // Implementation, returns y
        }                                       // Closes method
}                                               // Closes class
```

To invoke this method from another program (or class), the following call could be included in the program:

```
int a = java.lang.Math.max (2, 3);
```

Upon reaching this statement, the computer would go and find the max method under the Math class in the java.lang package, input "2" and "3" as arguments, and then return a "3," which would then be set as the value of "a."

The above example illustrates a point critical to our first main copyright issue, namely that the declaration line beginning "public static" is entirely dictated by the rules of the language. In order to declare a particular *functionality*, the language *demands* that the method declaration take a particular form. There is no choice in how to express it. To be specific, that line reads:

```
public static int max (int x, int y) {
```

The word "public" means that other programs can call on it. (If this instead says "private," then it can only be accessed by other methods inside the same class.) The word "static" means that the method can be invoked without creating an instance of the class. (If this instead is an instance method, then it would always be invoked with respect to an object.) The word "int" means that an integer is returned by the method. (Other alternatives are "boolean," "char," and "String" which respectively mean "true/false," "single character," and "character string.") Each of these three parameters is drawn from a short menu of possibilities, each possibility corresponding to a very specific functionality. The word "max" is a name and while any name (other than a reserved word) could have been used, names themselves cannot be copyrighted, as will be shown. The phrase "(int x, int y)" identifies the arguments that must be passed into the method, stating that they will be in integer form. The "x" and the "y" could be "a" and "b" or "arg1" and "arg2," so there is a degree of creativity in naming the arguments. Again, names cannot be copyrighted. (Android did not copy all of the particular argument names used in Java but did so as to some arguments.) Finally, "{" is the beginning marker that tells the compiler that the method body is about to follow. The marker is mandatory. The foregoing description concerns the rules for the language itself. Again, each parameter choice other than the names has a precise functional choice. If someone wants to implement a particular function, the declaration specification can only be written in one way.

Part of the declaration of a method can list any exceptions. When a program violates the semantic constraints of the Java language, the Java virtual machine will signal this error to the program as an exception for special handling. These are specified via "throw" statements appended at the end of a declaration. Android and Java are not identical in their throw designations but they are very similar as to the 37 packages at issue.

A Java program must have at least one class. A typical program would have more than one method in a class. Packages are convenient folders to organize the classes.

This brings us to the application programming interface. When Java was first introduced in 1996, the API included eight packages of pre-written programs. At least three of these packages were "core" packages, according to Sun, fundamental to being able to use the Java language at all. These packages were java.lang, java.io, and java.util. As a practical matter, anyone free to use the language itself (as Oracle concedes all are), must also use the three core packages in order to make any worthwhile use of the language. Contrary to Oracle, there is no bright line between the language and the API.

Each package was broken into classes and those in turn broken into methods. For example, java.lang (a package) included Math (a class) which in turn included max (a method) to return the greater of two inputs, which was (and remains) callable as java.lang.Math.max with appropriate arguments (inputs) in the precise form required (see the example above).

After Java's introduction in 1996, Sun and the Java Community Process, a mechanism for developing a standard specifications for Java classes and methods, wrote hundreds more programs to carry out various nifty functions and they were organized into coherent packages by Sun to become the Java application programming interface. In 2008, as stated, the Java API had grown from the original eight to 166 packages with over six hundred classes with over six thousand methods. All of it was downloadable from Sun's (now Oracle's) website and usable by anyone, including Java application developers, upon agreement to certain license restrictions. Java was particularly useful for writing programs for use via the Internet and desktop computers.

Although the declarations must be the same to achieve the same functionality, the names of the methods and the way in which the methods are grouped do not have to be the same. Put differently, many different API organizations could supply the same overall range of functionality. They would not, however, be interoperable. Specifically, code written for one API would not run on an API organized differently, for the name structure itself dictates the precise form of command to call up any given method.

To write a fresh program, a programmer names a new class and adds fields and methods. These methods can call upon the pre-written functions in the API. Instead of re-inventing the wheels in the API from scratch, programmers can call on the tried-and-true pre-packaged programs in the API. These are ready-made to perform a vast menu of functions. This is the whole point of the API. For example, a student in high school can write a program that can call upon java.lang.Math.max to return the greater of two numbers, or to find the cosine of an angle, as one step in a larger homework assignment. Users and developers can supplement the API with their own specialized methods and classes.

The foregoing completes the facts necessary to decide the copyrightability issue but since Oracle has made much of two small items copied by Google, this order will now make findings thereon so that there will be proper context for the court of appeals.

3. RANGECHECK AND THE DE-COMPILED TEST FILES.

Oracle has made much of nine lines of code that crept into both Android and Java. This circumstance is so innocuous and overblown by Oracle that the actual facts, as found herein by the judge, will be set forth below for the benefit of the court of appeals.

Dr. Joshua Bloch worked at Sun from August 1996 through July 2004, eventually holding the title of distinguished engineer. While working at Sun, Dr. Bloch wrote a nine-line code for a function called “rangeCheck,” which was put into a larger file, “Arrays.java,” which was part of the class library for the 37 API packages at issue. The function of rangeCheck was to check the range of a list of values before sorting the list. This was a very simple function.

In 2004, Dr. Bloch left Sun to work at Google, where he came to be the “chief Java architect” and “Java guru.” Around 2007, Dr. Bloch wrote the files, “Timsort.java” and “ComparableTimsort,” both of which included the same rangeCheck function he wrote while at Sun. He wrote the Timsort files in his own spare time and not as part of any Google project. He planned to contribute Timsort and ComparableTimsort back to the Java community by submitting his code to an open implementation of the Java platform, OpenJDK, which was controlled by Sun. Dr. Bloch did, in fact, contribute his Timsort file to OpenJDK and Sun included Timsort as part of its Java J2SE 5.0 release.

In 2009, Dr. Bloch worked on Google’s Android project for approximately one year. While working on the Android team, Dr. Bloch also contributed Timsort and ComparableTimsort to the Android platform. Thus, the nine-line rangeCheck function was copied into Google’s Android. This was how the infringement happened to occur.

When discovered, the rangeCheck lines were taken out of the then-current version of Android over a year ago. The rangeCheck block of code appeared in a class containing 3,179 lines of code. This was an innocent and inconsequential instance of copying in the context of a massive number of lines of code.

Since the remainder of this order addresses only the issue concerning structure, sequence and organization, and since rangeCheck has nothing to do with that issue, rangeCheck will not be mentioned again, but the reader will please remember that it has been readily conceded that these nine lines of code found their way into an early version of Android.

Google also copied eight computer files by decompiling the bytecode from eight Java files back into source code and then using the source code. These files were merely used as test files and never found their way into Android or any handset. These eight files have been treated at trial as a single unit.

Line by line, Oracle tested all fifteen million lines of code in Android (and all files used to test along the way leading up to the final Android) and these minor items were the only items copied, save and except for the declarations and calls which, as stated, can only be written in one way to achieve the specified functionality.

ANALYSIS AND CONCLUSIONS OF LAW

1. NAMES AND SHORT PHRASES.

To start with a clear-cut rule, names, titles and short phrases are not copyrightable, according to the United States Copyright Office, whose rule thereon states as follows:

> Copyright law does not protect names, titles, or short phrases or expressions. Even if a name, title, or short phrase is novel or distinctive or lends itself to a play on words, it cannot be protected by copyright. The Copyright Office cannot register claims to exclusive rights in brief combinations of words such as:
>
> - Names of products or services.
> - Names of business organizations, or groups (including the names of performing groups).
> - Pseudonyms of individuals (including pen or stage names).
> - Titles of works.

- Catchwords, catchphrases, mottoes, slogans, or short advertising expressions.
- Listings of ingredients, as in recipes, labels, or formulas. When a recipe or formula is accompanied by an explanation or directions, the text directions may be copyrightable, but the recipe or formula itself remains uncopyrightable.

U.S. Copyright Office, Circular 34; *see* 37 C.F.R. 202.1(a).

This rule is followed in the Ninth Circuit. *Sega Enters., Ltd. v. Accolade, Inc.*, 977 F.2d 1510, 1524 n.7 (9th Cir. 1992). This has relevance to Oracle's claim of copyright ownership over names of methods, classes and packages.

2. THE DEVELOPMENT OF LAW ON THE COPYRIGHTABILITY OF COMPUTER PROGRAMS AND THEIR STRUCTURE, SEQUENCE AND ORGANIZATION.

Turning now to the more difficult question, this trial showcases a distinction between copyright protection and patent protection. It is an important distinction, for copyright exclusivity lasts 95 years whereas patent exclusivity lasts twenty years. And, the Patent and Trademark Office examines applications for anticipation and obviousness before allowance whereas the Copyright Office does not. This distinction looms large where, as here, the vast majority of the code was *not* copied and the copyright owner must resort to alleging that the accused stole the "structure, sequence and organization" of the work. This phrase — structure, sequence and organization — does not appear in the Act or its legislative history. It is a phrase that crept into use to describe a residual property right where literal copying was absent. A question then arises whether the copyright holder is more appropriately asserting an exclusive right to a functional system, process, or method of operation that belongs in the realm of patents, not copyrights.

A. *Baker v. Seldon.*

The general question predates computers. In the Supreme Court's decision in *Baker v. Seldon*, 101 U.S. 99 (1879), the work at issue was a book on a new system of double-entry bookkeeping. It included blank forms, consisting of ruled lines, and headings, illustrating the system. The accused infringer copied the method of bookkeeping but used different forms. The Supreme Court framed the issue as follows:

> The evidence of the complainant is principally directed to the object of showing that Baker uses the same system as that which is explained and illustrated in Selden's books. It becomes important, therefore, to determine whether, in obtaining the copyright of his books, he secured the exclusive right to the use of the system or method of book-keeping which the said books are intended to illustrate and explain.

Id. at 101. *Baker* held that using the same accounting system would not constitute copyright infringement. The Supreme Court explained that only patent law can give an exclusive right to a method:

> To give to the author of the book an exclusive property in the art described therein, when no examination of its novelty has ever been officially made, would be a surprise and a fraud upon the public. That is the province of letters-patent, not of copyright. The claim to an invention or discovery of an art or manufacture must be subjected to the examination of the Patent Office before an exclusive right therein can be obtained; and it can only be secured by a patent from the government.

Id. at 102. The Supreme Court went on to explain that protecting the method under copyright law would frustrate the very purpose of publication:

> The copyright of a work on mathematical science cannot give to the author an exclusive right to the methods of operation which he propounds, or to the diagrams which he employs to explain them, so as to prevent an engineer from using them whenever occasion requires. The very object of publishing a book on science or the useful arts is to communicate to the world the useful knowledge which it contains. But this object would be frustrated if the knowledge could not be used without incurring the guilt of piracy of the book.

Id. at 103. *Baker* also established the "merger" doctrine for systems and methods intermingled with the texts or diagrams illustrating them:

> And where the art it teaches cannot be used without employing the methods and diagrams used to illustrate the book, or such as are similar to them, such methods and diagrams are to be considered as necessary incidents to the art, and given therewith to the public; not given for the purpose of publication in other works explanatory of the art, but for the purpose of practical application.

Ibid. It is true that *Baker* is aged but it is not passé. To the contrary, even in our modern era, *Baker* continues to be followed in the appellate courts, as will be seen below.

B. The Computer Age and Section 102(b) of the 1976 Act.

Almost a century later, Congress revamped the Copyright Act in 1976. By then, software for computers was just emerging as a copyright issue. Congress decided in the 1976 Act that computer programs would be copyrightable as "literary works." *See* H.R. REP. NO. 94-1476, at 54 (1976). There was, however, no express definition of a computer program until an amendment in 1980.

The 1976 Act also codified a *Baker*-like limitation on the scope of copyright protection in Section 102(b). *See Apple Computer, Inc. v. Microsoft Corp.*, 35 F.3d 1435, 1443 n.11 (9th Cir. 1994). Section 102(b) stated (and still states):

> In no case does copyright protection for an original work of authorship extend to any idea, procedure, process, system, method of operation, concept, principle, or discovery, regardless of the form in which it is described, explained, illustrated, or embodied in such work.

The House Report that accompanied Section 102(b) of the Copyright Act explained:

> Copyright does not preclude others from using the ideas or information revealed by the author's work. It pertains to the literary, musical, graphic, or artistic form in which the author expressed intellectual concepts. Section 102(b) makes clear that copyright protection does not extend to any idea, procedure, process, system, method of operation, concept, principle, or discovery, regardless of the form in which it is described, explained, illustrated, or embodied in such work.
>
> *Some concern has been expressed lest copyright in computer programs should extend protection to the methodology or processes adopted by the programmer, rather than merely to the 'writing' expressing his ideas. Section 102(b) is intended, among other things, to make clear that the expression adopted by the programmer is the copyrightable element in a computer program, and that the actual processes or methods embodied in the program are not within the scope of the copyright law.*
>
> Section 102(b) in no way enlarges or contracts the scope of copyright protection under the present law. Its purpose is to restate, in the context of the new single Federal system of copyright, that the basic dichotomy between expression and idea remains unchanged.

H.R. REP. NO. 94-1476, at 56–57 (1976) (emphasis added).[5]

Recognizing that computer programs posed novel copyright issues, Congress established the National Commission on New Technological Uses of Copyrighted Works (referred to as CONTU) to recommend the extent of copyright protection for software. The Commission consisted of twelve members with Judge Stanley Fuld as chairman and Professor Melville Nimmer as vice-chairman.

The Commission recommended that a definition of "computer program" be added to the copyright statutes. This definition was adopted in 1980 and remains in the current statute:

> A "computer program" is a set of statements or instructions to be used directly or indirectly in a computer in order to bring about a certain result.

17 U.S.C. 101. Moreover, the CONTU report stated that Section 102(b)'s preclusion of copyright protection for "procedure, process, system, method of operation" was reconcilable with the new definition of "computer program." The Commission explained the dichotomy between copyrightability and non-copyrightability as follows:

> Copyright, therefore, protects the program so long as it remains fixed in a tangible medium of expression but does not protect the electromechanical functioning of a machine. The way copyright affects games and game-playing is closely analogous: one may not adopt and republish or redistribute copyrighted game rules, but the copyright owner has no power to prevent others from playing the game.
>
> *Thus, one is always free to make a machine perform any conceivable process (in the absence of a patent), but one is not free to take another's program.*

NAT'L COMM'N ON NEW TECHNOLOGICAL USES OF COPYRIGHTED WORKS, FINAL REPORT 20 (1979) (emphasis added). The Commission also recognized the "merger" doctrine, a rule of importance a few pages below in this order (emphasis added):

> The "idea-expression identity" exception provides that copyrighted language may be copied without infringing when there is but a limited number of ways to express a given idea. This rule is the logical extension of the fundamental principle that copyright cannot protect ideas. *In the computer context this means that when*

[5] The Court has reviewed the entire legislative history. The quoted material above is the only passage of relevance. This order includes a summary of the CONTU report but it came after-the-fact and had little impact on the Act other than to include a definition of "computer program."

> *specific instructions, even though previously copyrighted, are the only and essential means of accomplishing a given task, their later use by another will not amount to an infringement . . . .* [C]opyright protection for programs does not threaten to block the use of ideas or program language previously developed by others when that use is necessary to achieve a certain result. When other language is available, programmers are free to read copyrighted programs and use the ideas embodied in them in preparing their own works.

Ibid. The Commission realized that differentiating between the copyrightable form of a program and the uncopyrightable process was difficult, and expressly decided to leave the line drawing to federal courts:

> [T]he many ways in which programs are now used and the new applications which advancing technology will supply may make drawing the line of demarcation more and more difficult. To attempt to establish such a line in this report written in 1978 would be futile. . . . Should a line need to be drawn to exclude certain manifestations of programs from copyright, that line should be drawn on a case-by-case basis by the institution designed to make fine distinctions — the federal judiciary.

Id. at 22–23.

Congress prepared no legislative reports discussing the CONTU comments regarding Section 102(b). *See* H.R. REP. NO. 96-1307, at 23–24 (1980). Nevertheless, Congress followed CONTU's recommendations by adding the definition of computer programs to the statute and amending a section of the Act not relevant to this order. *See Apple Computer, Inc. v. Formula Intern. Inc.*, 725 F.2d 521, 522–25 (9th Cir. 1984).

Everyone agrees that no one can copy line-for-line someone else's copyrighted computer program. When the line-by-line listings are different, however, some copyright owners have nonetheless accused others of stealing the "structure, sequence and organization" of the copyrighted work. That is the claim here.

C. Decisions Outside the Ninth Circuit.

No court of appeals has addressed the copyrightability of APIs, much less their structure, sequence and organization. Nor has any district court. Nevertheless, a review of the case law regarding non-literal copying of software provides guidance. Circuit decisions outside the Ninth Circuit will be considered first.

The Third Circuit led off in *Whelan Associates, Inc. v. Jaslow Dental Laboratory, Inc.*, 797 F.2d 1222 (3d Cir. 1986). In that case, the claimant owned a program, Dentalab, that handled the administrative and bookkeeping tasks of dental prosthetics businesses. The accused infringer developed another program, Dentcom, using a different programming language. The Dentcom program handled the same tasks as the Dentalab program and had the following similarities:

> The programs were similar in three significant respects . . . most of the file structures, and the screen outputs, of the programs were virtually identical . . . five particularly important "subroutines" within both programs — order entry, invoicing, accounts receivable, end of day procedure, and end of month procedure — performed almost identically in both programs.

Id. at 1228. On these facts, the district court had found, after a bench trial, that the accused infringer copied the claimant's software program. *Id.* at 1228–29.

On appeal, the accused infringer argued that the structure of the claimant's program was not protectable under copyright. In rejecting this argument, the court of appeals created the following framework to deal with non-literal copying of software:

> [T]he line between idea and expression may be drawn with reference to the end sought to be achieved by the work in question. In other words, *the purpose or function of a utilitarian work would be the work's idea, and everything that is not necessary to that purpose or function would be part of the expression of the idea*.

Id. at 1236 (emphasis in original). Applying this test, *Whelan* found that the structure of Dentalab was copyrightable because there were many different ways to structure a program that managed a dental laboratory:

> [T]he idea of the Dentalab program was the efficient management of a dental laboratory (which presumably has significantly different requirements from those of other businesses). Because that idea could be accomplished in a number of different ways with a number of different structures, the structure of the Dentalab program is part of the program's expression, not its idea.

Id. at 1236 n.28. The phrase "structure, sequence and organization" originated in a passage in *Whelan* explaining that the opinion used those words interchangeably and that, although not themselves part of the Act, they were intended to capture the thought that "sequence and order could be parts of the expression, not the idea, of a work." *Id.* at 1239, 1248.

To summarize, in affirming the district court's final judgment of infringement, *Whelan* held that the *structure* of the Dentalab program was copyrightable because there were many other ways to perform the same function of handling the administrative and bookkeeping tasks of dental prosthetics businesses with different structures and designs. *Id.* at 1238. Others were free to come up with their own version but could not appropriate the Dentalab structure. This decision plainly seems to have been the high-water mark of copyright protection for the structure, sequence and organization of computer programs. It was also the only appellate decision found by the undersigned judge that affirmed (or directed) a final judgment of copyrightability on a structure, sequence and organization theory.

Perhaps because it was the first appellate decision to wade into this problem, *Whelan* has since been criticized by subsequent treatises, articles, and courts, including our own court of appeals. *See Sega Enters., Ltd. v. Accolade, Inc.*, 977 F.2d 1510, 1524–25 (9th Cir. 1992). Instead, most circuits, including ours, have adopted some variation of an approach taken later by the Second Circuit. *See Apple Computer, Inc. v. Microsoft Corp.*, 35 F.3d 1435, 1445 (9th Cir. 1994).

In *Computer Associates International, Inc. v. Altai*, 982 F.2d 693 (2d Cir. 1992), the claimant owned a program designed to translate the language of another program into the particular language that the computer's operating system would be able to understand. The accused infringer developed its own program with substantially similar structure but different source code (using the same programming language). The Second Circuit criticized *Whelan* for taking too narrow a view of the "idea" of a program. The Second Circuit adopted instead an "abstract-filtration-comparison" test. The test first dissected the copyrighted program into its structural components:

> In ascertaining substantial similarity under [the abstract-filtration-comparison test], a court would first break down the allegedly infringed program into its constituent structural parts. Then, by examining each of these parts for such things as incorporated ideas, expression that is necessarily incidental to those ideas, and elements that are taken from the public domain, a court would then be able to sift out all non-protectable material.

Id. at 706.

Then, the test filtered out structures that were not copyrightable. For this filtration step, the court of appeals relied on the premise that programmers fashioned structures "to maximize the program's speed, efficiency, as well as simplicity for user operation, while taking into consideration certain externalities such as the memory constraints of the computer upon which the program will be run." *Id.* at 698. Because these were "practical considerations," the court held that structures based on these considerations were not copyrightable expressions.

Thus, for the filtration step, the court of appeals outlined three types of structures that should be precluded from copyright protection. *First*, copyright protection did not extend to structures dictated by efficiency. A court must inquire

> whether the use of *this particular set* of modules [is] necessary efficiently to implement that part of the program's process being implemented. If the answer is yes, then the expression represented by the programmer's choice of a specific module or group of modules has merged with their underlying idea and is unprotected.

Id. at 708 (emphasis in original). Paradoxically, this meant that non-efficient structures might be copyrightable while efficient structures may not be. Nevertheless, the Second Circuit explained its reasoning as follows:

> In the context of computer program design, the concept of efficiency is akin to deriving the most concise logical proof or formulating the most succinct mathematical computation. Thus, the more efficient a set of modules are, the more closely they approximate the idea or process embodied in that particular aspect of the program's structure.
>
> While, hypothetically, there might be a myriad of ways in which a programmer may effectuate certain functions within a program — *i.e.*, express the idea embodied in a given subroutine — efficiency concerns may so narrow the practical range of choice as to make only one or two forms of expression workable options.

Ibid. Efficiency also encompassed user simplicity and ease of use. *Id.* at 708–09.

Second, copyright protection did not extend to structures dictated by external factors. The court explained this as follows:

> [I]n many instances it is virtually impossible to write a program to perform particular functions in a specific computing environment without employing standard techniques. This is a result of the fact that a programmer's freedom of design choice is often circumscribed by extrinsic considerations such as (1) the mechanical specifications of the computer on which a particular

> program is intended to run; (2) compatibility requirements of other programs with which a program is designed to operate in conjunction; (3) computer manufacturers' design standards; (4) demands of the industry being serviced; and (5) widely accepted programming practices within the computer industry.

Id. at 709–10.

Third, copyright protection did not extend to structures already found in the public domain. The court reasoned that materials in the public domain, such as elements of a computer program that have been freely accessible, cannot be appropriated. *Ibid.* Ultimately, in the case before it, the Second Circuit held that after removing unprotectable elements using the criteria discussed above, only a few lists and macros in accused product were similar to the copied product, and their impact on the program was not large enough to declare copyright infringement. *Id.* at 714–15. The copyright claim, in short, failed.

The Tenth Circuit elaborated on the abstract-filtration-comparison test in *Gates Rubber Co. v. Bando Chemical Industries, Ltd.*, 9 F.3d 823 (10th Cir. 1993). There, the claimant developed a computer program that determined the proper rubber belt for a particular machine by performing complicated calculations involving numerous variables. The program used published formulas in conjunction with certain mathematical constants developed by the claimant to determine belt size. The Tenth Circuit offered the following description of a software program's structure:

> The program's architecture or structure is a description of how the program operates in terms of its various functions, which are performed by discrete modules, and how each of these modules interact with each other.

Id. at 835. As had the Second Circuit, the Tenth Circuit held that filtration should eliminate the unprotectable elements of processes, facts, public domain information, merger material, *scenes a faire* material, and other unprotectable elements suggested by the particular facts of the program under examination. For Section 102(b) processes, the court gave the following description:

> Returning then to our levels of abstraction framework, we note that processes can be found at any level, except perhaps the main purpose level of abstraction. Most commonly, processes will be found as part of the system architecture, as operations within modules, or as algorithms.

Id. at 837. The court described the *scenes a faire* doctrine for computer programs as follows:

> The *scenes a faire* doctrine also excludes from protection those elements of a program that have been dictated by external factors. In the area of computer programs these external factors may include: hardware standards and mechanical specifications, software standards and compatibility requirements, *Sega Enterprises Ltd. v. Accolade, Inc.*, 977 F.2d 1510, 1525–27 (9th Cir. 1993), computer manufacturer design standards, target industry practices and demands, and computer industry programming practices.
>
> *　　　*　　　*
>
> We recognize that the *scenes a faire* doctrine may implicate the protectability of interfacing and that this topic is very sensitive and has the potential to effect [sic] widely the law of computer copyright. This appeal does not require us to determine the scope of the *scenes a faire* doctrine as it relates to interfacing and accordingly we refrain from discussing the issue.

Id. at 838 & n.14 (all citations omitted except *Sega*). Like the Second Circuit, the Tenth Circuit also listed many external considerations — such as compatibility, computer industry programming practices, and target industry practices and demands — that would exclude elements from copyright protection under the *scenes a faire* doctrine. Ultimately, the Tenth Circuit remanded because the district court had failed to make specific findings that fit this framework.

The First Circuit weighed in with its 1995 decision *Lotus Development Corp. v. Borland International, Inc.*, 49 F.3d 807 (1st Cir. 1995). In *Lotus*, the claimant owned the Lotus 1-2-3 spreadsheet program that enabled users to perform accounting functions electronically on a computer. Users manipulated and controlled the program via a series of menu commands, such as “Copy,” “Print,” and “Quit.” In all, Lotus 1-2-3 had 469 commands arranged into more than 50 menus and submenus. Lotus 1-2-3 also allowed users to write “macros,” whereby a user could designate a series of command choices (sequence of menus and submenus) with a single macro keystroke. Then, to execute that series of commands, the user only needed to type the single pre-programmed macro keystroke, causing the program to recall and perform the designated series of commands automatically. *Id.* at 809–10.

The accused infringer Borland developed a competing spreadsheet program. Borland included the Lotus menu command hierarchy in its program to make it compatible with Lotus 1-2-3 so that spreadsheet users who were already familiar with Lotus 1-2-3 would be able to switch to the Borland program without having to learn new commands or rewrite their Lotus macros. In so doing, Borland did not copy any of Lotus's underlying source or object code. (The opinion did not say whether the programs were written in the same language.)

The district court had ruled that the Lotus 1-2-3 menu command hierarchy was a copyrightable expression because there were many ways to construct a spreadsheet menu tree. Thus, the district court had concluded that the Lotus developers' choice and arrangement of command terms, reflected in the Lotus menu command hierarchy, constituted copyrightable expression. *Id.* at 810–11.

The First Circuit, however, held that the Lotus menu command hierarchy was not copyrightable because it was a method of operation under Section 102(b). The court explained:

> We think that "method of operation," as that term is used in § 102(b), refers to the means by which a person operates something, whether it be a car, a food processor, or a computer. Thus a text describing how to operate something would not extend copyright protection to the method of operation itself; other people would be free to employ that method and to describe it in their own words. Similarly, if a new method of operation is used rather than described, other people would still be free to employ or describe that method.

Id. at 815.

The court reasoned that because the menu command hierarchy was essential to make use of the program's functional capabilities, it should be properly categorized as a "method of operation" under Section 102(b). The court explained:

> The Lotus menu command hierarchy does not merely explain and present Lotus 1-2-3's functional capabilities to the user; it also serves as the method by which the program is operated and controlled . . . . In other words, to offer the same capabilities as Lotus 1-2-3, Borland did not have to copy Lotus's underlying code (and indeed it did not); to allow users to operate its programs in substantially the same way, however, Borland had to copy the Lotus menu command hierarchy. Thus the Lotus 1-2-3 code is not a uncopyrightable "method of operation."

Ibid. Thus, the court reasoned that although Lotus had made "expressive" choices of what to name the command terms and how to structure their hierarchy, it was nevertheless an uncopyrightable "method of operation." The *Lotus* decision was affirmed by an evenly divided Supreme Court (four to four).

The Federal Circuit had the opportunity to apply *Lotus* in an appeal originating from the District of Massachusetts in *Hutchins v. Zoll Medical Corp.*, 492 F.3d 1377 (Fed. Cir. 2007) (affirming summary judgment against copyright owner). In *Hutchins*, the claimant owned a program for performing CPR and argued that his copyright covered the "system of logic whereby CPR instructions are provided by computerized display, and [] the unique logic contained in [his] software program." *Id.* at 1384. The claimant argued that the accused program was similar because it "perform[ed] the same task in the same way, that is, by measuring heart activity and signaling the quantity and timing of CPR compressions to be performed by the rescuer." *Ibid.* The court of appeals rejected this argument, holding that copyright did not protect the "technologic method of treating victims by using CPR and instructing how to use CPR." *Ibid.* (citing *Lotus*).

D. Decisions in the Supreme Court and in our Circuit.

Our case is governed by the law in the Ninth Circuit and, of course, the Supreme Court. The Supreme Court missed the opportunity to address these issues in *Lotus* due to the four-to-four affirmance and has, thus, never reached the general question. Nonetheless, *Baker*, which is still good law, provides guidance and informs how we should read Section 102(b).

Another Supreme Court decision, *Feist Publications, Inc. v. Rural Telephone Services Co., Inc.*, 499 U.S. 340 (1991), which dealt primarily with the copyrightability of purely factual compilations, provided some general principles. In *Feist*, the Supreme Court considered the copyrightability of a telephone directory comprised of names, addresses, and phone numbers organized in alphabetical order. The Supreme Court rejected the notion that copyright law was meant to reward authors for the "sweat of the brow." This meant that we should not yield to the temptation to award copyright protection merely because a lot of sweat went into the work. The Supreme Court concluded that protection only extended to the original components of an author's work. *Id.* at 353. The Supreme Court concluded:

> This inevitably means that the copyright in a factual compilation is thin. Notwithstanding a valid copyright, a subsequent compiler

> remains free to use the facts contained in another's publication to aid in preparing a competing work, so long as the competing work does not feature the same selection and arrangement.

Id. at 349.

Turning to our own Ninth Circuit, our court of appeals has recognized that non-literal components of a program, including the structure, sequence and organization and user interface, can be protectable under copyright depending on whether the structure, sequence and organization in question qualifies as an expression of an idea rather than an idea itself. *Johnson Controls, Inc. v. Phoenix Control Sys., Inc.*, 886 F.2d 1173, 1175 (9th Cir. 1989). This decision arrived between the Third Circuit's *Whelan* decision and the Second Circuit's *Computer Associates* decision. *Johnson Controls* is one of Oracle's mainstays herein.

In *Johnson Controls*, the claimant developed a system of computer programs to control wastewater treatment plants. The district court found that the structure, sequence and organization of the program was expression and granted a preliminary injunction even though the accused product did not have similar source or object code. *Id.* at 1174. Therefore, the standard of review on appeal was limited to abuse of discretion and clear error. Our court of appeals affirmed the preliminary injunction, stating that the claimant's program was very sophisticated and each individual application was customized to the needs of the purchaser, indicating there may have been room for individualized expression in the accomplishment of common functions. Since there was some discretion and opportunity for creativity in the structure, the structure of the program was expression rather than an idea. *Id.* at 1175. *Johnson Controls*, however, did not elaborate on which particular structures deserved copyright protection.

In *Brown Bag Software v. Symantec Corp.*, 960 F.2d 1465 (9th Cir. 1992), our court of appeals outlined a two-part test for determining similarity between computer programs: the extrinsic and intrinsic tests. This pertained to infringement, not copyrightability. The claimant, who owned a computer program for outlining, alleged that an accused infringer copied his program's non-literal features. *Id.* at 1472. The claimant alleged that seventeen specific features in the programs were similar. On summary judgment, the district court had found that each feature was either not protectable or not similar as a matter of law:

> The district court ruled that one group of features represented a claim of copyright in "concepts . . . fundamental to a host of computer programs" such as "the need to access existing files, edit the work, and print the work." As such, these features, which took the form of four options in the programs' opening menus, were held to be unprotectable under copyright.
>
> A second group of features involved "nine functions listed in the menu bar" and the fact that "virtually all of the functions of the PC-Outline program [ ] can be performed by Grandview." The district court declared that "these functions constitute the idea of the outlining program" and, furthermore, "[t]he expression of the ideas inherent in the features are . . . distinct." The court also held that "the similarity of using the main editing screen to enter and edit data . . . is essential to the very idea of a computer outlining program."
>
> The third group of features common to PC-Outline and Grandview concerned "the use of pull-down windows." Regarding these features, the district court made three separate rulings. The court first found that "[p]laintiffs may not claim copyright protection of an . . . expression that is, if not standard, then commonplace in the computer software industry" . . . . [and] that the pull-down windows of the two programs look different.

Id. at 1472–73. Our court of appeals affirmed the district court's order without elaborating on the copyrightability rulings quoted above.

In *Atari Games Corp. v. Nintendo of America Inc.*, 975 F.2d 832 (Fed. Cir. 1992), the Federal Circuit had occasion to interpret Ninth Circuit copyright precedent. In *Atari*, the claimant Nintendo sued Atari for copying the Nintendo 10NES program, which prevented the Nintendo game console from accepting unauthorized game cartridges. Atari deciphered the 10NES program through reverse engineering and developed its own program to unlock the Nintendo game console. Atari's new program generated signals indistinguishable from 10NES but was written in a different programming language. *Id.* at 835–36.

Applying our Ninth Circuit precedents, *Johnson Controls* and *Brown Bag*, the Federal Circuit affirmed the district court's preliminary injunction for copyright infringement.

The Federal Circuit held that the 10NES program contained copyrightable expression because it had organization and sequencing unnecessary to the unlocking function:

> Nintendo's 10NES program contains more than an idea or expression necessarily incident to an idea. Nintendo incorporated within the 10NES program creative organization and sequencing *unnecessary* to the lock and key function. Nintendo chose arbitrary programming instructions and arranged them in a unique sequence to create a purely arbitrary data stream. This data stream serves as the key to unlock the NES. Nintendo may protect this creative element of the 10NES under copyright.

Id. at 840 (emphasis added). The Federal Circuit stated that there were creative elements in the 10NES program

> beyond the literal expression used to effect the unlocking process. The district court defined the unprotectable 10NES idea or process as the generation of a data stream to unlock a console. This court discerns no clear error in the district court's conclusion. The unique arrangement of computer program expression which generates that data stream does not merge with the process so long as alternate expressions are available. In this case, Nintendo has produced expert testimony showing a multitude of different ways to generate a data stream which unlocks the NES console.

Ibid. (citation omitted). Thus, the Federal Circuit held that the district court did not err in concluding that the 10NES program contained protectable expression and affirmed the preliminary injunction.

Next came two decisions holding that Section 102(b) bars from copyright software interfaces necessary for interoperability. The Section 102(b) holdings arose in the context of larger holdings that it had been fair use to copy software to reverse-engineer it so as to isolate the unprotectable segments. These two decisions will now be described in detail.

In *Sega Enterprises Ltd. v. Accolade, Inc.*, 977 F.2d 1510 (9th Cir. 1992), the accused infringer had to copy object code in order to understand the interface procedures between the Sega game console and a game cartridge, that is, how the software in the game console interacted with the software in the game cartridge to achieve compatibility. *Id.* at 1515–16. After learning and documenting these interactions (interface procedures), the accused infringer wrote its own source code to mimic those same interface procedures in its own game cartridges so that its cartridges could run on the Sega console. Our court of appeals held that the copying of object code for the purpose of achieving compatibility was fair use. Notably, in its fair-use analysis, our court of appeals *expressly held that the interface procedures for compatibility were functional aspects not copyrightable under Section 102(b)*: "Accolade copied Sega's software solely in order to discover the functional requirements for compatibility with the Genesis console — aspects of Sega's programs that are not protected by copyright. 17 U.S.C. § 102(b)." *Id.* at 1522. The court used the phrase "interface procedures," a term describing the interface between applications, multiple times to describe the functional aspect of the interaction between software programs and summarized its analysis of copyrightability as follows:

> In summary, the record clearly establishes that disassembly of the object code in Sega's video game cartridges was necessary in order to understand the functional requirements for Genesis compatibility. The *interface procedures* for the Genesis console are distributed for public use only in object code form, and are not visible to the user during operation of the video game program. Because object code cannot be read by humans, it must be disassembled, either by hand or by machine. Disassembly of object code necessarily entails copying. Those facts dictate our analysis of the second statutory fair use factor. If disassembly of copyrighted object code is per se an unfair use, the owner of the copyright gains a de facto monopoly over *the functional aspects of his work — aspects that were expressly denied copyright protection by Congress.* 17 U.S.C. § 102(b). In order to enjoy a lawful monopoly over the idea or functional principle underlying a work, the creator of the work must satisfy the more stringent standards imposed by the patent laws. *Bonito Boats, Inc. v. Thunder Craft Boats, Inc.*, 489 U.S. 141, 159–64, 109 S.Ct. 971, 982–84, 103 L.Ed.2d 118 (1989). Sega does not hold a patent on the Genesis console.

Sega, 977 F.2d at 1526 (emphasis added). In *Sega*, the interface procedure that was required for compatibility was "20 bytes of initialization code plus the letters S–E–G–A." *Id.* at 1524 n.7. Our court of appeals found that this interface procedure was functional and therefore not copyrightable under Section 102(b). The accused infringer Accolade was free to copy this interface procedure for use in its own games to ensure compatibility with the Sega Genesis game console. Our court of appeals distinguished the *Atari* decision, where the Federal Circuit had found that the Nintendo's 10NES security system was infringed, because there was only one signal that unlocked the Sega console, unlike the "multitude of different ways to unlock" the Nintendo console:

> We therefore reject Sega's belated suggestion that Accolade's incorporation of the code which "unlocks" the Genesis III console is not a fair use. Our decision on this point is entirely consistent

> with *Atari v. Nintendo*, 975 F.2d 832 (Fed. Cir.1992). Although *Nintendo* extended copyright protection to Nintendo's 10NES security system, that system consisted of an original program which generates an arbitrary data stream "key" which unlocks the NES console. Creativity and originality went into the design of that program. *See id.* at 840. Moreover, the federal circuit concluded that there is a "multitude of different ways to generate a data stream which unlocks the NES console." *Atari*, 975 F.2d at 839. The circumstances are clearly different here. Sega's key appears to be functional. It consists merely of 20 bytes of initialization code plus the letters S–E–G–A. There is no showing that there is a multitude of different ways to unlock the Genesis III console.

Sega, 977 F.2d at 1524 n.7.

This order reads *Sega* footnote seven (quoted above) as drawing a line between copying functional aspects necessary for compatibility (not copyrightable) versus copying functional aspects unnecessary for compatibility (possibly copyrightable). Our court of appeals explained that in *Atari*, the Nintendo game console's 10NES program had had functionality *unnecessary* to the lock-and-key function. *See also Atari*, 975 F.2d at 840. Since the accused infringer Atari had copied the entire 10NES program, it also had copied aspects of the 10NES program unnecessary for compatibility between the console and game cartridges. This was inapposite to the facts of *Sega*, where the accused infringer Accolade's final product duplicated *only* the aspect of Sega's program *necessary* for compatibility between the console and game cartridges. Thus, the holding of our court of appeals was that the aspect of a program necessary for compatibility was unprotectable, specifically invoking Section 102(b), but copyrightable expression could still exist for aspects unnecessary for compatibility.

The *Sega* decision and its compatibility reasoning was followed in a subsequent reverse-engineering decision by our court of appeals, *Sony Computer Entertainment, Inc., v. Connectix Corporation*, 203 F.3d 596 (9th Cir. 2000). The facts were somewhat different in *Sony*. There, the accused infringer Connectix did not create its own games for Sony's Playstation game console; instead, the accused infringer created an emulated environment that duplicated the interface procedures of Sony's console so that games written for Sony's console could be played on a desktop computer running the emulator. In order to do this, the accused infringer copied object code for the Sony Playstation's operating software, its BIOS program, in order to discover signals sent between the BIOS and the rest of the game console. *Id.* at 600. After uncovering these signals (again, application interfaces), the accused infringer wrote its own source code to *duplicate these interfaces* in order to create its emulator for the desktop computer. Thus, games written for the Playstation console were playable on Connectix's emulator for the desktop computer. Citing Section 102(b) and *Sega*, our court of appeals stated that the Playstation BIOS contained "unprotected functional elements," and concluded that the accused infringer's intermediate step of copying object code was fair use because it was done for the "purpose of gaining access to the unprotected elements of Sony's software." *Id.* at 602–03.[6]

* * *

With apology for its length, the above summary of the development of the law reveals a trajectory in which enthusiasm for protection of "structure, sequence and organization" peaked in the 1980s, most notably in the Third Circuit's *Whelan* decision. That phrase has not been re-used by the Ninth Circuit since *Johnson Controls* in 1989, a decision affirming preliminary injunction. Since then, the trend of the copyright decisions has been more cautious. This trend has been driven by fidelity to Section 102(b) and recognition of the danger of conferring a monopoly by copyright over what Congress expressly warned should be conferred only by patent. This is not to say that infringement of the structure, sequence and organization is a dead letter. To the contrary, it is not a dead letter. It is to say that the *Whelan* approach has given way to the *Computer Associates* approach, including in our own circuit. *See Sega Enters., Ltd. v. Accolade, Inc.*, 977 F.2d 1510, 1525 (9th Cir. 1992); *Apple Computer, Inc. v. Microsoft Corp.*, 35 F.3d 1435, 1445 (9th Cir. 1994).

In this connection, since the CONTU report was issued in 1980, the number of software patents in force in the United States has dramatically increased from barely a thousand in 1980 to hundreds of thousands today. *See* Iain Cockburn, *Patents, Tickets and the Financing*

---

[6] *Sega* and *Sony* are not the only Ninth Circuit decisions placing a premium on functionality as indicating uncopyrightability. Other such decisions were surveyed in the summary earlier in this order. *See also Triad Sys. Corp. v. Southeastern Exp. Co.*, 64 F.3d 1330, 1336 (9th Cir. 1995); *Apple Computer, Inc. v. Microsoft Corp.*, 35 F.3d 1435, 1444 (9th Cir. 1994); *Apple Computer, Inc. v. Formula Intern., Inc.*, 725 F.2d 521, 525 (9th Cir. 1984).

of Early-Stage Firms: Evidence from the Software Industry, 18 JOURNAL OF ECONOMICS & MANAGEMENT STRATEGY 729–73 (2009). This has caused at least one noted commentator to observe:

> As software patents gain increasingly broad protection, whatever reasons there once were for broad copyright protection of computer programs disappear. Much of what has been considered the copyrightable "structure, sequence and organization" of a computer program will become a mere incident to the patentable idea of the program or of one of its potentially patentable subroutines.

Mark Lemley, *Convergence in the Law of Software Copyright?*, 10 HIGH TECHNOLOGY LAW JOURNAL 1, 26–27 (1995). Both Oracle and Sun have applied for and received patents that claim aspects of the Java API. *See, e.g.*, U.S. Patents 6,598,093 and 7,006,855. (These were not asserted at trial.)[7]

* * *

In view of the foregoing, this order concludes that our immediate case is controlled by these principles of copyright law:

- Under the merger doctrine, when there is only one (or only a few) ways to express something, then no one can claim ownership of such expression by copyright.
- Under the names doctrine, names and short phrases are not copyrightable.
- Under Section 102(b), copyright protection never extends to any idea, procedure, process, system, method of operation or concept

[7] The issue has been debated in the journals. For example, Professor Pamela Samuelson has argued that Section 102(b) codified the *Baker* exclusion of procedures, processes, systems, and methods of operation for computer programs as well as the pre-*Baker* exclusion of high-level abstractions such as ideas, concepts, and principles. Pamela Samuelson, *Why Copyright Law Excludes Systems and Processes from the Scope of Protection*, 85 TEX. L. REV. 1921 (2007). In contrast, Professor David Nimmer (the son of Professor Melville Nimmer) has argued that Section 102(b) should not deny copyright protection to "the expression" of a work even if that work happens to consist of an idea, procedure or process. 1-2 NIMMER ON COPYRIGHT § 2.03[D] (internal citations omitted). Similarly, Professor Jane Ginsburg has argued that the Section 102(b) terms "process," "system," and "method of operation" should not be understood literally for computer programs. Jane Ginsburg, *Four Reasons and a Paradox: The Manifest Superiority of Copyright Over Sui Generis Protection of Computer Software*, 94 COLUM. L. REV. 2559, 2569–70 (1994).

regardless of its form. Functional elements essential for interoperability are not copyrightable.

- Under *Feist*, we should not yield to the temptation to find copyrightability merely to reward an investment made in a body of intellectual property.

APPLICATION OF CONTROLLING LAW TO CONTROLLING FACTS

All agree that everyone was and remains free to program in the Java language itself. All agree that Google was free to use the Java language to write its own API. While Google took care to provide fresh line-by-line implementations (the 97 percent), it generally replicated the overall name organization and functionality of 37 packages in the Java API (the three percent). The main issue addressed herein is whether this violated the Copyright Act and more fundamentally whether the replicated elements were copyrightable in the first place.

This leads to the first holding central to this order and it concerns the method level. The reader will remember that a method is like a subroutine and over six thousand are in play in this proceeding. As long as the specific code written to implement a method is different, anyone is free under the Copyright Act to write his or her own method to carry out exactly the same function or specification of any and all methods used in the Java API. Contrary to Oracle, copyright law does not confer ownership over any and all ways to implement a function or specification, no matter how creative the copyrighted implementation or specification may be. The Act confers ownership only over the specific way in which the author wrote out his version. Others are free to write their own implementation to accomplish the identical function, for, importantly, ideas, concepts and functions cannot be monopolized by copyright.

To return to our example, one method in the Java API carries out the function of comparing two numbers and returning the greater. Google — and everyone else in the world — was and remains free to write its own code to carry out the identical function so long as the implementing code in the method body is different from the copyrighted implementation. This is a simple example, but even if a method resembles higher mathematics, everyone is still free to try their hand at writing a different implementation, meaning that they are free to use the same inputs to derive the same outputs (while throwing the same exceptions) so long as the implementation in between is their own. The House Report, quoted above, stated in 1976 that "the actual processes or methods embodied in the program are not within the scope of the copyright law." H.R. REP. NO. 94-1476, at 57 (1976).

Much of Oracle's evidence at trial went to show that the design of methods in an API was a creative endeavor. Of course, that is true. Inventing a new method to deliver a new output can be creative, even inventive, including the choices of inputs needed and outputs returned. The same is true for classes. But such inventions — at the concept and functionality level — are protectable only under the Patent Act. The Patent and Trademark Office examines such inventions for validity and if the patent is allowed, it lasts for twenty years. Based on a single implementation, Oracle would bypass this entire patent scheme and claim ownership over any and all ways to carry out methods for 95 years — without any vetting by the Copyright Office of the type required for patents. This order holds that, under the Copyright Act, no matter how creative or imaginative a Java method specification may be, the entire world is entitled to use the same method specification (inputs, outputs, parameters) so long as the line-by-line implementations are different. To repeat the Second Circuit's phrasing, "there might be a myriad of ways in which a programmer may . . . express the idea embodied in a given subroutine." *Computer Associates*, 982 F.2d at 708. The method specification is the *idea*. The method implementation is the *expression*. No one may monopolize the *idea*.[8]

To carry out any given function, the method specification as set forth in the declaration *must be identical* under the Java rules (save only for the choices of argument names). Any other declaration would carry out some *other* function. The declaration requires precision. Significantly, when there is only one way to write something, the merger doctrine bars anyone from claiming exclusive copyright ownership of that expression. Therefore, there can be no copyright violation in using the identical declarations. Nor can there be any copyright violation

[8] Each method has a singular purpose or function, and so, the basic function or purpose of a method will be an unprotectable process. *Gates Rubber Co. v. Bando Chemical Industries, Ltd.*, 9 F.3d 823, 836 (10th Cir. 1993); *see Apple Computer, Inc. v. Formula Intern. Inc.*, 725 F.2d 521, 525 (9th Cir. 1984) (holding that while a particular set of instructions is copyrightable, the underlying computer process is not).

due to the *name* given to the method (or to the arguments), for under the law, names and short phrases cannot be copyrighted.

In sum, Google and the public were and remain free to write their own implementations to carry out exactly the same functions of all methods in question, using exactly the same method specifications and names. Therefore, at the method level — the level where the heavy lifting is done — Google has violated no copyright, it being undisputed that Google's implementations are different.

As for classes, the rules of the language likewise insist on giving names to classes and the rules insist on strict syntax and punctuation in the lines of code that declare a class. As with methods, for any desired functionality, the declaration line will *always* read the same (otherwise the functionality would be different) — save only for the name, which cannot be claimed by copyright. Therefore, under the law, the declaration line cannot be protected by copyright. This analysis is parallel to the analysis for methods. This now accounts for virtually all of the three percent of similar code.

* * *

Even so, the second major copyright question is whether Google was and remains free to group its methods in the same way as in Java, that is, to organize its Android methods under the same class and package scheme as in Java. For example, the Math classes in both systems have a method that returns a cosine, another method that returns the larger of two numbers, and yet another method that returns logarithmic values, and so on. As Oracle notes, the rules of Java did not insist that these methods be grouped together in any particular class. Google could have placed its trigonometric function (or any other function) under a class other than Math class. Oracle is entirely correct that the rules of the Java language did not require that the same grouping pattern (or even that they be grouped at all, for each method could have been placed in a stand-alone class).[9]

---

[9] As to the groupings of methods within a class, Google invokes the *scenes a faire* doctrine. That is, Google contends that the groupings would be so expected and customary as to be permissible under the *scenes a faire* doctrine. For example, the methods included under the Math class are typical of what one would expect to see in a group of math methods. Just as one would expect certain items in the alcove for nuts, bolts and screws

Oracle's best argument, therefore, is that while no single name is copyrightable, Java's overall system of organized names — covering 37 packages, with over six hundred classes, with over six thousand methods — is a "taxonomy" and, therefore, copyrightable under *American Dental Association v. Delta Dental Plans Association*, 126 F.3d 977 (7th Cir. 1997). There was nothing in the rules of the Java language that required that Google replicate the same groupings even if Google was free to replicate the same functionality.[10]

The main answer to this argument is that while the overall scheme of file name organization resembles a taxonomy, it is *also* a command structure for a system or method of operation of the application programming interface. The commands are (and must be) in the form

java.package.Class.method()

and each calls into action a pre-assigned function.[11]

To repeat, Section 102(b) states that "in no case does copyright protection for an original work of authorship extend to any idea, procedure, process, system, method of operation . . . regardless of the form . . . ." That a system or method of operation has thousands of commands arranged in a creative taxonomy does not change its character as a method of operation. Yes, it is creative. Yes, it is original. Yes, it resembles a taxonomy. But it is nevertheless a command structure, a system or method of operation — a long hierarchy of

in a hardware store, one would expect the methods of the math class to be in, say, a typical math class. At trial, however, neither side presented evidence from which we can now say that the same is true for all the other hundreds of classes at issue. Therefore, it is impossible to say on this record that *all* of the classes and their contents are typical of such classes and, on this record, this order rejects Google's global argument based on *scenes a faire*.

[10] This is a good place to point out that while the groupings appear to be the same, when we drill down into the detail code listings, we see that the actual sequences of methods in the listings are different. That is, the sequence of methods in the class Math in Android is different from the sequence in the same class in Java, although all of the methods in the Java version can be found somewhere in the Android version, at least as shown in their respective listings (TX 47.101, TX 623.101). The Court has not compared all six-hundred-plus classes. Nor has any witness or counsel so far on the record. Oracle does not, however, contend that the actual sequences would track method-for-method and it has not so proven. This detailed observation, however, does not change the fact that all of the methods in the Java version can be found somewhere in the Android version, classified under the same classes.

[11] The parentheses indicate that inputs/arguments may be included in the command.

over six thousand commands to carry out pre-assigned functions. For that reason, it cannot receive copyright protection — patent protection perhaps — but not copyright protection.

* * *

Interoperability sheds further light on the character of the command structure as a system or method of operation. Surely, millions of lines of code had been written in Java before Android arrived. These programs necessarily used the java.package.Class.method() command format. These programs called on all or some of the specific 37 packages at issue and necessarily used the command structure of names at issue. Such code was owned by the developers themselves, not by Oracle. *In order for at least some of this code to run on Android, Google was required to provide the same java.package.Class.method() command system using the same names with the same "taxonomy" and with the same functional specifications.* Google replicated what was necessary to achieve a degree of interoperability — but no more, taking care, as said before, to provide its own implementations.

That interoperability is at the heart of the command structure is illustrated by Oracle's preoccupation with what it calls "fragmentation," meaning the problem of having imperfect interoperability among platforms. When this occurs, Java-based applications may not run on the incompatible platforms. For example, Java-based code using the replicated parts of the 37 API packages will run on Android but will not if a 38th package is needed. Such imperfect interoperability leads to a "fragmentation" — a Balkanization — of platforms, a circumstance which Sun and Oracle have tried to curb via their licensing programs. In this litigation, Oracle has made much of this problem, at times almost leaving the impression that if only Google had replicated *all* 166 Java API packages, Oracle would not have sued. While fragmentation is a legitimate business consideration, it begs the question whether or not a license was required in the first place to replicate some or all of the command structure. (This is especially so inasmuch as Android has not carried the Java trademark, and Google has not held out Android as fully compatible.) The immediate point is this: fragmentation, imperfect interoperability, and Oracle's angst over it illustrate the character of the command structure as a functional system or method of operation.

In this regard, the Ninth Circuit decisions in *Sega* and *Sony*, although not on all fours, are close analogies. Under these two decisions, interface procedures required for interoperability were deemed "functional requirements for compatibility" and were not copyrightable under Section 102(b). Both decisions held that interface procedures that were necessary to duplicate in order to achieve interoperability were functional aspects not copyrightable under Section 102(b). Here, the command structure for the 37 packages (including inheritances and exception throws), when replicated, at least allows interoperability of code using the replicated commands. To the extent of the 37 packages — which, after all, is the extent of Oracle's copyright claim — *Sega* and *Sony* are analogous. Put differently, if someone could duplicate the interfaces of the Sony BIOS in order to run the Playstation games on desktops (taking care to write its own implementations), then Google was free to duplicate the command structure for the 37 packages in Android in order to accommodate third-party source code relying on the 37 packages (taking care to write its own implementations). Contrary to Oracle, "full compatibility" is not relevant to the Section 102(b) analysis. In *Sony*, the accused product implemented only 137 of the Playstation BIOS's 242 functions because those were the only functions invoked by the games tested. Connectix's Opening Appellate Brief at 18, available at 1999 WL 33623860, (9th Cir. May 27, 1999). Our court of appeals held that the accused product "itself infringe[d] no copyright." *Sony*, 203 F.3d at 608 n.11. This parallels Google's decision to implement some but not all of the Java API packages in Android.

* * *

This explains why *American Dental Association v. Delta Dental Plans Association*, 126 F.3d 977 (7th Cir. 1997), is not controlling. Assuming arguendo that a taxonomy is protectable by copyright in our circuit, *see Practice Mgmt. Info. Corp. v. Am. Med. Ass'n*, 121 F.3d 516 (9th Cir. 1997), the taxonomy in *ADA* had nothing to do with computer programs. It was not a system of commands, much less a system of commands for a computer language. The taxonomy there subdivided the universe of all dental procedures into an outline of numbered categories with English-language descriptions created by the ADA. This was then to be used by insurance companies and dentists to facilitate billings. By contrast, here the taxonomy is composed entirely of a system of commands to carry out specified computer functions. For a similar reason, Oracle's analogy to stealing the plot and character from a movie is inapt, for movies involve no "system" or "method of operation" — scripts are entirely creative.

In *ADA*, Judge Frank Easterbrook (writing for the panel) suggested that a "system" under Section 102(b) had to come with "instructions for use." 126 F.3d at 980. Because the taxonomy there at issue had no instructions for use, among other reasons, it was held not to be a system. By contrast, the API at issue here does come with instructions for use, namely, the documentation and embedded comments that were much litigated at trial. They describe every package, class and method, what inputs they need, and what outputs they return — the classic form of instructions for use.

In our circuit, the structure, sequence and organization of a computer program may (or may not) qualify as a protectable element depending on the "particular facts of each case" and always subject to exclusion of unprotectable elements. *Johnson Controls v. Phoenix Control Sys.*, 886 F.2d 1173, 1175 (9th Cir. 1989). Contrary to Oracle, *Johnson Controls* did not hold that all structure, sequence and organization in all computer programs are within the protection of a copyright. On a motion for preliminary injunction, the district court found that the structure, sequence and organization of the copyrighted program, on the facts there found, deserved copyright protection. (The structure, sequence and organization features found protectable were not described in the appellate decision.) On an appeal from the preliminary injunction, our court of appeals merely said no clear error had occurred. Again, the appellate opinion stated that the extent to which the structure, sequence and organization was protectable depended on the facts and circumstances of each case. The circumstances there are not the circumstances here.

In closing, it is important to step back and take in the breadth of Oracle's claim. Of the 166 Java packages, 129 were not violated in any way. Of the 37 accused, 97 percent of the Android lines were new from Google and the remaining three percent were freely replicable under the merger and names doctrines. Oracle must resort, therefore, to claiming that it owns, by copyright, the exclusive right to any and all possible implementations of the taxonomy-like command structure for the 166 packages and/or any subpart thereof — even though it copyrighted only one implementation. To accept Oracle's claim would be to allow anyone to copyright one version of code to carry out a system of commands and thereby bar all others from writing their own different versions to carry out all or part of the same commands. No holding has ever endorsed such a sweeping proposition.

CONCLUSION

This order does not hold that Java API packages are free for all to use without license. It does not hold that the structure, sequence and organization of all computer programs may be stolen. Rather, it holds on the specific facts of this case, the particular elements replicated by Google were free for all to use under the Copyright Act. Therefore, Oracle's claim based on Google's copying of the 37 API packages, including their structure, sequence and organization is **DISMISSED**. To the extent stated herein, Google's Rule 50 motions regarding copyrightability are **GRANTED** (Dkt. Nos. 984, 1007). Google's motion for a new trial on copyright infringement is **DENIED AS MOOT** (Dkt. No. 1105).

IT IS SO ORDERED.

Dated: May 31, 2012.

[illegible]

WILLIAM ALSUP
UNITED STATES DISTRICT JUDGE